基于交织预取率的帮助线程预取质量调节算法 *

张建勋 [1]，古志民 [2]

(1. 天津职业技术师范大学 信息技术工程学院，天津 300222; 2. 北京理工大学 计算机学院，北京 100081)

摘　要：预执行帮助线程在预取过程中需要进行动态预取调节，而传统静态枚举控制参数值的控制方法在预取执行过程中保持固定不变，从而使得该方法不能够有效的为主线程提供预取质量保证（quality of service，QoS）。针对该问题，提出了一种基于交织预取率的帮助线程预取质量参数调节方法。首先，对帮助线程的预取 QoS 优化进行了建模分析；其次，在前期交织预取工作的基础上，提出了基于交织预取率的帮助线程参数值调节算法；最后，在真实的商用多核平台上对所提出帮助线程预取调节算法进行了评测和分析。实验结果是所提出的帮助线程预取调节算法使得基准测试程序的几何平均性能加速比为 1.114，而传统静态枚举方法的几何平均性能加速比为 1.135。实验结果表明，所提出的帮助线程预取质量调节算法解决了帮助线程预取过程中的参数值自动调节问题，算法不需静态枚举参数值便可以快速获得与之相近似的预取性能提升。

关键词：预取率；帮助线程；预取 QoS；动态调节

中图分类号：TP391　　doi: 10.3969/j.issn.1001-3695.2017.09.0877

Helper thread prefetching quality adjust algorithm based on prefetching rate

Xing Mingming[1], Xing Mingming[2], Xing Ming[3]

(1. *College of Information Technology & Engineering*, *Tianjin University of Technology and Education*, *Tianjin* 300222, *China*; 2. *College of Computer Science*, *Beijing Institute of Technology*, *Beijing* 100081, *China*)

Abstract: Pre-execution helper threads were needed to perform dynamic prefetching in the prefetch process. However, the traditional static enumeration parameter value was keep constant during in the execution of prefetching, so that the method can not effectively provide prefetching Quality. To solve this problem, a method of prefetching quality control based on prefetching rate was proposed. Firstly, the model of helper thread prefetching QoS optimization is analyzed. Secondly, based on the previous work, this paper puts forward the algorithm of parameter value adjustment based on prefetching rate. In the end, the algorithm is evaluated and analyzed on the real commercial multi-core platform. Experimental results were as follows. With the proposed prefetching adjusting algorithm, the geometric average performance speed up of the benchmarks was 1.114. It was similar to the traditional static enumeration method's geometric average performance speed up which was 1.135. The experimental results show that the proposed algorithm can solve the helper thread prefetching quality adjust problem, as well can quickly obtain the similar prefetching performance compared with the static enumeration method.

Key Words: prefetch rate; helper thread; prefetch Qos; dynamical adjust

0　引言

随着云计算和大数据技术的发展，高性能计算系统的存储性能已经成为制约大数据应用处理性能提高的重要瓶颈。大数据应用程序在算法执行过程中需要大量的数据访问操作，其数据访问模式呈现出非规则和非一致性特征，致使传统的软件硬件预取方案在这种应用场景下失效。针对非规则访存行为的大数据应用程序，现有的工作提出采用基于多参数控制的预执行帮助线程预取技术[1]来隐藏长延迟指令的访存延迟。基于预执行思想的帮助线程技术能够有效的将主线程所需要的数据从主存中推送到多核平台的最后一级缓存中，将程序的非连续局部性转换为算法局部性，从而提高应用程序的性能。然而通过手工枚举选择控制参数值的过程是一项繁杂耗时的工作，因此实现帮助线程控制参数值的实时选择和自适应调节是解决上述问题的关键。

本文在前期工作的基础上[2-6]，围绕帮助线程预取控制策略

基金项目：国家自然科学基金资助项目（61070029，61370062）；天津市高等学校科技发展基金计划项目（JWK1618）；天津职业技术师范大学科研启动基金（KYQD1619）

作者简介：张建勋（1978-），男，河北顺平人，副教授，博士，主要研究方向为多核缓存优化（zhangjx@tute.edu.cn）；古志民（1964-2016），男，教授，博导，主要研究方向为并行计算与系统结构、多核计算、缓存优化等.

的自适应调节展开研究工作。针对帮助线程预取执行过程中的实时控制问题，提出一个基于预取率的帮助线程参数调节算法，并在多核平台上对所提出的算法进行了系统评测。实验结果表明，提出的帮助线程预取参数动态调节算法无须采用手动枚举参数值即可获得与手动枚方法相近似的性能提升。

1 相关工作

随着多核处理器的广泛普及，基于线程级的前瞻执行技术引起了学术界的广泛关注[7,8]。硬件实现方法需要额外的更改硬件来实时的产生指令执行的动态片断。软件实现方法主要在源码级和二级制级对串行程序进行优化，这些软件实现方案需要离线的剖析程序执行语义信息。Kim 等人[9]首次将帮助线程预执行技术引入到真实的 Intel 同时多线程平台，针对线程之间的同步问题，提出了两种机制一是基于循环的同步，另外一种是基于采样的同步。Song 等人[10]提出了一个编译框架用于帮助线程代码的自动构造。Jung 等人[11]提出一种基于 PV 信号的帮助线程同步机制。Kamruzzaman 等人[12]提出一种用于核际预取方案，帮助线程在空闲核执行数据预取，在执行一段距离之后，帮助线程和主线程互换 CPU 处理核，主线程迁移到帮助线程所在的 CPU 处理核享受帮助线程预取带来的获益，帮助线程迁移到主线程所在处理核继续进行数据预取操作。该方案是将正在执行的程序划分成等长的块，块的大小预先通过程序的离线剖析来设定。Lu 等人[13]利用 ADORE 动态优化框架实现帮助线程数据预取的目的。Luo 等人[14]提出一种对前瞻线程的动态性能调节技术，通过实时监测硬件性能计数器对的带有前瞻线程执行时的程序进行性能监测，并实时评估出原有串行程序的执行时间进而判断前瞻执行线程是否有效，根据评估结果对前瞻线程作出不同的决策。Fei 等人[15]在交织预取帮助线程的基础上提出了一种梯度优化帮助线程控制参数的方法。Andrew 等人[16,17]针对多核平台的缓存主存等共享资源的使用提出一种基于 PID（proportion integration differentiation，PID）控制调节思想的应用程序 QoS 技术。该技术是一种基于 PID 的控制机制，通过硬件性能计数器实时监测应用程序对共享资源的使用情况，然后通过降低主频、时钟调制等方法限制应用程序对共享资源的使用，从而为运行在多核平台上不同类型的应用程序提供 QoS 保证。Luo 和 Andrew 的工作对实现帮助线程预取 QoS 的动态调节提供了可供借鉴的技术和思想。

综上所述，传统帮助线程研究对于帮助线程的控制参数值都是通过对串行程序进行离线剖析之后凭经验设定。这种人工选取参数值的过程需要凭借经验，并且工作量巨大，不大适用于实际应用。本文将就交织预取帮助线程的预取控制参数 K，P 和 B 的取值在线取值优选和调节开展研究工作，实现实时自适应参数值选择的调节算法。

2 帮助线程预取 QoS 优化模型

帮助线程引入预取距离 K、预取大小 P 和同步块数量 B 来实现对帮助线程预取行为的调节和控制。为了便于描述，下面给出帮助线程预取 QoS 策略优化问题的形式化描述。

假设 Q 是一个包含帮助线程预取 QoS 策略的集合，Q=｛Q_i｝,i=1,2,3…m。其中 Q_i=（K，P，B）是一个三元组，K，P，B 分别代表交织预取帮助线程的三个控制参数取值，本文称 Q_i 为一个帮助线程预取 QoS 策略。其中 K 表示帮助线程领先主线程的预取距离，以热点循环数目表示；P 表示帮助线程开始为主线程预取的工作量，也是以循环数表示；B 表示帮助线程多长时间与主线程同步一次，同样以循环数表示，同步的主要目的是避免帮助线程偏离主线程的执行路径，即落后或过度领先于主线程的执行。

定义 1 交织预取率 R_p。帮助线程中预取率定义为一个 Block 数据块内帮助线程预取工作量大小所占的比例，即 $R_p = P/(K+P), R_p \in (0,1)$。所谓交织预取是指当主线程进入热点循环后，为了避免帮助线程滞后于主线程，帮助线程并不是预取所有的热点数据，而是跳跃性的为主线程预取数据，而留下一部分热点数据的访问由主线程来访问，从而使得主线程和帮助线程共同分担长延迟访存指令，二者的访存数据流相互交织，并行执行，因此称为交织预取。

从预取率的定义可以看出，当预取距离已知时，可以通过预取率 R_p 调节帮助线程预取工作量的大小。

定义 2 预取 QoS 评价函数。

$$\theta_i = Eval(Q_i, S_i) = IPC_i (1 \le i \le m)$$

评价函数的输入参数有两个，一个是预取 QoS 策略 Q_i，另外一个是热点模块执行样本 S_i。IPC_i 表示第 i 个采样周期内应用预取 QoS 策略 Q_i 后所得到的主线程性能指标。

假设主线程热点模块 Γ 是一个指令流，Γ=｛$i_1,i_2,i_3,\ldots i_N$｝。U 是热点模块执行样本大小，在此以循环数表示。将程序热点模块的指令流 Γ 划分成长度等于 U 的指令序列，即 Γ=｛S_i｝i=0,1,2…,L，其中｜Si｜=U，S_i 为热点模块采样执行样本。

帮助线程预取 QoS 优化可以表示为

$$Max\{Eval(Q_i, S_i)\}(1 \le i \le m)$$

即在热点模块的指令流中，通过选择长度为 U 的指令序列为热点模块采样执行样本，在程序执行过程中对每一个热点模块执行样本 S_i 采用相应的预取 QoS 策略 Q_i，采样结束后通过评价函数计算该采样样本的 CPI_i，最后选择出具有最小 CPI 值的交织预取 QoS 策略。

由定义 2 可知，$\theta_i = Eval(Q_i, S_i) = IPC_i$，$\theta_i$ 是对热点模块执行样本 S_i 的性能评测，θ_i 值越大，说明主线程所在处理器执行每条指令的时钟数越低，主线程的性能越好。

定义 3 设 θ_i 是预取调节框架对采样样本 S_i 的性能评测结果，θ_j 是预取调节框架对采样样本 S_j 的性能评测结果，则 $\varepsilon_{i,j} = \theta_j - \theta_i$ 称为从预取 QoS 策略 Q_i 到预取 QoS 策略 Q_j 时主线程性能的改变，其中 *i=1,2,…，j=1,2,…*。$\varepsilon_{i,j}$ 表示不同的预取 QoS 策略对两个热点模块采样样本性能提升的差别。若 $\varepsilon_{i,j}$ >0，则表示预取 QoS 策略 Q_j 优先于预取 QoS 策略 Q_i；若 $\varepsilon_{i,j}$ ≈0，

表示两个预取 QoS 策略对主线程性能提升能力相同；若 $\varepsilon_{i,j}<0$，则表示预取 QoS 策略 Q_i 优先于预取 QoS 策略 Q_j。

定义 4　热点模块采样本从 S_0 到 S_c 分别采用预取 QoS 策略 Q_0 到 Q_c 之后，如果 $\varepsilon_{0,1}>0,\varepsilon_{1,2}>0,\ldots,\varepsilon_{i,c}>0$，主线程性能提升具有改进趋势；相反，若 $\varepsilon_{0,1}<0,\varepsilon_{1,2}<0,\ldots,\varepsilon_{i,c}<0$，表示主线程性能提升具有恶化趋势。其中称 c 为性能渐变趋势阈值。

3　基于预取率的帮助线程调节算法

帮助线程预取调节的实质是通过调整帮助线程的预取工作量来实现控制帮助线程预取行为的目的。预取距离 *K* 和预取工作量 *P* 是两个相关联的变量，二者的关系可以用预取率 R_p 来描述。在既定的预取距离下，总能找到一个恰当的 R_p 值使得主线程的性能相对最优。传统的分别调节单个变量的枚举方法使得算法的收敛速度很慢，不能够快速找到相对局部最优的参数值组合。本节将介绍一个基于预取率 R_p 的帮助线程预取调节算法，将即定范围内的参数值根据预取率 R_p 快速的遍历生成一次，然后将生成的参数值组合作为一个预取 *QoS* 策略应用于帮助线程，在参数训练期对该策略的性能进行评价。

3.1　确定参数值上限

热点模块中热循环的外层循环次数限定了 KPB 参数的最大取值。假设热循环的外层循环次数是 L_{max}，则 K，P 和 B 取值满足下列约束条件：

a) $0\leq K\leq L_{max}$，$0\leq P\leq L_{max}-K$，$0\leq B*(K+P)\leq L_{max}$

b) $0\leq B\leq (L_{max}/(K+P))$

针对交织预取帮助线程，Huang[18,19]等人提出了一种基于缓存相关度的预取距离评估策略，主要用于评估预取距离 K 的取值范围。缓存相关度是将热点循环映射到多核平台最后一级共享缓存的组相联度来评估预取距离的一个概念。其核心思想是对于一个热点循环，如果执行到第 j 次循环后，热点循环中的数据已经将缓存冲满，如果再继续访问热点循环的数据将会引发缓存替换，此时称热点循环相对于共享缓存的缓存相关度是 j。交织预取帮助线程的核心思想是由主线程与帮助线程的共同分担热点模块的 LLC 缓存缺失。

算法3-1: 基于预取率的帮助线程预取调节算法

```
输入:(K', Rp', CLAR, SYNC)
输出:(K,P)
init parameter K=K', Rp=Rp';
If (SYNC==1) then      //同步块数量为1
    B=1;
    if ( CLAR>0) then   //CLAR大于0表示主线程计算工作量足够
        K=0;
        if  (P in [0, P_upper limit]) then
            P=P+Step;
            Return K,P,B;
        endif
    else   //CLAR等于0，表示主线程计算工作量小
        if ( Rp in  (0, 1 ) ) then
            P=Rp*K/(1-Rp);
            Rp=Rp+0.1;          //调整预取比例
            return K,P,B;       //在预取率小于1之前只计算P，K不变
        else //预取率大于1表示当前K值下参数生成完毕
            if  (K in [0, K_upper limit]) then
                K=K+Step;    //调整K值
                Rp=0.1;      //重置预取率
                P=Rp*K/(1-Rp);
            endif
        endif
    Endif
Else // 同步块不为1，在当前已经找到的K和P基础上需要调整同步参数
    If (B < B_upper limit ) then
        B=B+1;
        return K,P,B;
    Endif
Endif
```

假设帮助线程最多可领先主线程 K_{max} 个循环，当主线程完成前 K_{max} 个循环的热点数据访问时，由于帮助线程相对于主线程代码量少，帮助线程至少也预取了 K_{max} 个循环的热点数据。如果此时 $2*K_{max}$ 超过了缓存相关度 j，那么帮助线程后续对热点数据的预取将可能导致缓存污染发生。因此 K 应该小于等于 j/2（取 K_{max}=j/2），此理论假设是帮助线程与主线程共享分担 LLC 缓存缺失指令，因此，此刻帮助线程的预取大小 P 是运行了 K_{max} 个热点数据，因此 P_{max}=j/2。

缓存相关度可以通过 Profiling 实验的方法得到。根据缓存相关度的思想，预取距离 K 和 P 还可以通过静态的方式预测。假设热点循环中每个热点循环所要访问的缓存缺失数据的大小为 $Loop_{size}$，LLC 共享缓存的容量 $Cache_{size}$，那么 K_{max}<=($Cache_{size}$/$Loop_{size}$)/2。静态预测的方法主要基于共享缓存的容量划分为两部分，分别由帮助线程和主线程共享，如果主线程中热点循环数 L 超过了($Cache_{size}$/$Loop_{size}$)/2 时缓存已经充满，此时帮助线程再向前预取数据会发生缓存污染，因此 K_{max} 和 P_{max} 参数值的上限设定为($Cache_{size}$/$Loop_{size}$)/2。

同步参数 B 表示同步距离，即多少个 Block 块同步一次，一般设置为 1，即每个 Block 块同步一次，参数 B 的最大上限值为（L_{max} /（K+P））。

3.2　基于预取率的帮助线程预取调节算法

帮助线程预取率 Rp 用来调整帮助线程预取行为，在既定 K 值下，通过遍历预取率 Rp 会大大缩短参数值的寻优时间。Rp 的取值范围是（0，1），预取大小 P 可以通过公式 $P=R_p*K/(1-R_p)$ 计算得出。参数值产生器的算法如算法 3-1 所示。

算法 3-1 中对于参数值的生成分为几种情况：

a）默认情况下参数 B=1，即每个块同步一次。

b）CYLR 标志着主线程计算工作量的大小，如果 CLAR>0 表示主线程具有足够工作量，在参数生成时取 K 值为 0，调整 P 值即可。

c）如果 CLAR 标志为 0 ，表示主线程工作量不足，需要同时调整参数 K 和 P。K 和 P 的调整是基于预取率的遍历。主要过程是先指定 K 值，然后计算当前 K 值下 Rp 值变化时生成相应的 P 值返回。之后，预取率重置为 0，K 值增加步长 STEP 为 K'，计算当前 K'值时，根据 Rp 值计算相应的 P 值返回。

d）SYNC 标志表示是否调节同步参数 B，一般同步块数量设置为 1，即 B=1。如果 SYNC 标志不为 1，表示需要在当前 K 和 P 的基础上增大同步距离，只调节参数 B。假设帮助线程预

取 QoS 策略个数用 Poilcy_num 表示，热点模块执行样本的大小为 U，每个采样样本用于评测一个预取 QoS 策略，因此帮助预取 QoS 策略训练期总样本大小为 Policy_num*U，训练期总样本越大，留给帮助预取预取优化可利用的空间就会变小，因此，预取参数调节算法要能够快速收敛，需要对生成参数值的数量做一定的限制。

4　实验评测与分析

4.1　实验平台与测试程序

在真实的商业机器上验证和评测帮助线程预取调节算法。实验平台使用 Intel Core 2 Quard Q6600 处理器，系统详细配置如表 1 所示。通过 Linux Affinity 接口将主线程和帮助线程绑定在固定的 CPU 处理核上。Q6600 的四个 CPU 处理核的硬件预取器处于打开状态。

基准测试程序采用了 7 个科学计算测试程序分别是 Mst、Em3d、Tsp、Health、SPEC CPU 2006 中的 Mcf、Gcc 和 Libquantum。表 2 是给出了测试程序的输入集和热点模块。表 2 的最后一列为预取帮助线程通过手动设置的最好参数值。除 426.Libquantum 之外，其他基准测试程序输入集均采用参考输入集。将动态参数调节与静态预取以及传统 PV[9]方法的帮助线程进行了比较。

表 1　Q6600 实验平台配置

Processor	Intel Core 2 Quad Processor Q6600
Memory	2 GB (DDR 667, non-ECC)
L1 D-Cache	32 KB*4 8 set-association cache line 64 bytes
L1 I-Cache	32 KB*4 8 set-association cache line 64 bytes
L2 Cache	4096 KB*2 16 set-association cache line 64 bytes
FSB Speed	1066 MHz
Compiler	Gcc version 4.3.0 -O2
OS	Fedora 9 with kernel 2.6.34

表 2　基准测试程序及预取控制参数值

基准测试程序	来源	数据输入集	热点模块	交织预取手动参数(*K–P–B*)
429.Mcf	SPEC2006	Ref/inp.in	Refresh_potential	24–24–nosyn
			Primal_bea_mpp	5–290–nosyn
403.Gcc	SPEC2006	Ref/166.i	Reg_is_remote_constant_p	192–50–nosyn
462. Libquantum	SPEC2006	365 26	quantum_cnot	8800–5800–1
			quantum_sigma_x	8800–5800–1
			quantum_toffoli	8800–5800–1
Mst	Olden	10000 1	HashLookup	90–45–1
Em3d	Olden	400000 128 75 1	Fill_from_fields	0–10–1
Tsp	Olden	10,000,000	Merge	160–50–1
Health	Olden	5 levels, 5000 iters	Check_patients_waiting	100–30–nosyn

表 3　程序热点模块被调用情况及参数值初值信息

程序热点模块	所属测试程序	被调用次数	预取距离 K 的上界	K 初值
fill_from_fields	em3d	2	50	2
HashLookup	mst	10000	3150	10
refresh_potential	mcf	36129	1500	10

4.2　实验结果与分析

4.2.1 帮助线程预取调节评测

为了消除程序运行过程中系统干扰因素，所有的测试程序运行了 7 次，取 7 次运行时间的中位数做为最终评测结果。图 1 所示是所有测试程序以原始串行程序为基准归一化后的执行时间，用 HPCF 代表所提出的预取调节框架。

与传统 PV 帮助线程方法相比，手动方式交织预取帮助线程方法效果最好，各个测试程序的平均性能加速比为 1.135，自动选择参数值的交织预取方法为 1.114，而传统 PV 方法的几何平均性能加速比为 1.020。采用自动选择参数值的方法，与手动选择参数值的方法性能比较接近，其性能差异在可接受的范围之内。与手动设置参数值方式的帮助线程相比，帮助线程预取的实时参数值调节的主要区别在于存在着参数值训练期和实时采样的开销。但是当问题的规模变大时，实时参数值选择方式和手动方式差距很小。例如 Mcf 的串行程序运行时间超过了 400 秒，通过实时采样动态调节帮助线程参数值方法使得主线程的加速比为 1.226，手动设置参数值的方法主线程的性能加速比为 1.232。

4.2.2 调节算法对预取的性能影响及分析

算法 3-1 所示的参数值调节算法主要思想是基于在既定 K 值下，基于预取率 Rp 计算帮助线程的预取大小 P。算法 3-1 中需要确定的信息主要包括预取距离 K 的上界 K_{max}、预取率 Rp 的步长以及参数值生成过程中 K 的增量步长 K_{step}。参数 K 的上限

值可以通过基于缓存相关度的性能剖析方法得到 K_{max} 的取值。K_{step} 决定着参数值产生器算法如何对参数 K 取值范围［0，K_{max}］进行划分，Rp 决定着预取距离与预取大小之间的比例关系，这两个变量的步长都决定着参数值产生器生成参数值的数量。

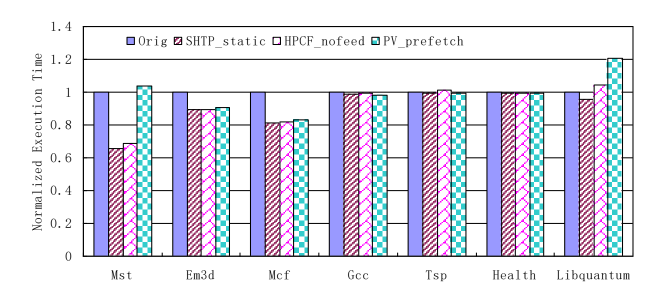


图 1　动态调节、静态选择参数与传统 PV 方法的预取性能比较

由于 Mcf、Mst 和 Em3d 三个测试程序应用交织预取时性能提升效果明显，因此本节将以这三个测试基准程序为例，详细分析 K_{step} 对预取调节框架性能的影响。基准测试程序运行 7 次，然后取 7 个数的中位数做为评测结果，以消除系统波动带来的影响。表 3 是三个基准测试程序热点模块被调用情况及参数 K 的初值及上限。预取调节框架参数值产生算法的执行过程是：将预取距离 K 的取值范围［0，K_{max}］以 K_{step} 为步长划分成等长的段，每一段选择一个 K 值，然后根据预取率 Rp 的变化产生参数 P。在初始实现的算法中 Rp 的取值是从 0.1 到 0.9，Rp 的步长选择为 0.1，即针对每个既定的预取距离 K 值生成 9 组（K，P）组合参数值。在此只讨论 K_{step} 步长对预取调节框架性能的影响。在 K 的初值 K_{init}、上限 K_{max} 和步长 K_{step} 给定的情况下，参数值产生器生成的参数值组合数量为 $P_{num}=((K_{max}-K_{init})/K_{step})\times 9$。

图 2~4 是 K_{step} 取不同步长时 Mst、Mcf 和 Em3d 程序应用不带反馈机制的预取调节框架后的性能加速比。

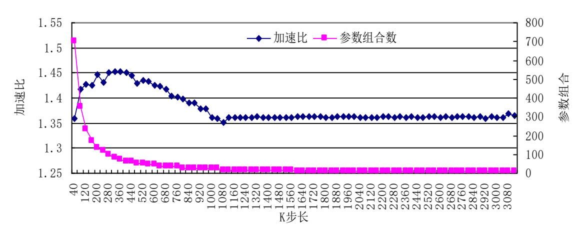


图 2　在不同 Kstep 步长时 Mst 程序性能加速比

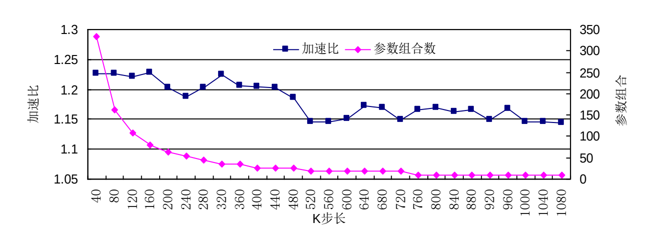


图 3　在不同 K_{step} 步长时 MCF 程序性能加速比

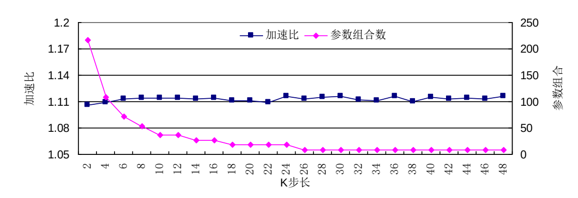


图 4　在不同 Kstep 步长时 Em3d 程序性能加速比

从三个图中可以看出，随着参数 K 步长的增大，预取调节产生的用于控制帮助线程的参数值组合数会逐步减少。当步长 K_{step} 增大到一定值后，参数值产生器在生成 K 值时，只会有一个 K 值，因为再增加步长就超出了 K 值的上限 K_{max}。例如图 2 中，Mst 程序中步长超过 1600 时，预取调节框架只是以（K_{init}+K_{step}）做为 K 值，通过变化 Rp 产生 9 组参数值做为候选参数值。同样，图 3 中，Mcf 程序中 K_{step} 超过 750 时，参数值产生器只会产生一个 K 值，相应 Rp 变化的参数值组合为 9 组。因此，K_{step} 步长不能选取过大，过大的步长导致候选参数值集变小，失去了得到局部最优参数值的机会。

三个测试程序中，除 Em3d 外，其余两个程序随着 K_{step} 的增大，程序性能加速比有下降的趋势，原因在于参数值生成器生成参数值组合数少，生成的参数值并不是局部最优参数值。但是当 K_{step} 过小时，会由于生成的参数值过多，从而增长参数值训练的时间，也会使程序性能下降。如图 2 中，Mst 程序在 K 值步长为 40 时，在 K 值的上限 3150 内会有 78 个 K 值分区段，相应的参数值产生器生成参数值组合数达到 702 组。假设在参数值初选阶段以热点模块的 1 次调用为采样样本的话，需要 702 次热点模块调用才能完成第一阶段参数值的训练，从而选择出 78 组参数值组合参与第二阶段参数值评估。在参数值复选阶段，如果增大采样样本的大小为 10 次热点模块调用的话，那么又需要 780 次采样样本训练。当步长 Kstep 为 40 时，总共需要 702+780=1410 次热点模块调用来训练参数值。而从表 3 中看到，Mst 程序的热点模块一共被调用了 10000 次，因此参数值训练期为程序热点总执行次数的 14.1%，因此程序的性能提升会因为步长较小时参数值训练期过长而受到影响。

5　结束语

帮助线程预取策略引入了预取距离 K、预取大小 P 和同步大小 B 三个控制参数，主要用于调节和控制帮助线程的预取行为。KPB 三参数的取值搜索空间通常很大，使用凭经验人工选优方法无法对如此庞大的搜索空间进行完整的搜索。本文提出了一种基于交织预取率的帮助线程的预取动态调节算法，算法能够实时在线完成帮助线程预取控制参数值的自适应调节。在 Intel Core 2 Quad Processor Q6600 处理器上通过对科学计算测试程序 Em3d、Mst 和 SPEC CPU benchmark 2006 中的 Mcf 进行实验。实验结果表明，提出的帮助线程自适应预取框架不需手工枚举参数值便可以快速获得与人工枚举方法相近似的性能提升。

参考文献：

[1] Luk C K. Tolerating memory latency through software-controlled pre-execution in simultaneous multithreading processors [C]// Proc of the 28th Annual International Symposium on Computer Architecture. New York: ACM Press, 2001: 40-51.

[2] Gu Z, Zheng N, Zhang Y, et al. The stable conditions of a task-pair with helper-thread in CMP [C]// Proc of International Conference on Parallel and Distributed Processing Techniques and Applications. 2009: 125-130.

[3] Gu Z, Fu Y, Zheng N, et al. Improving performance of the irregular data intensive application with small workload for CMPs [C]// Proc of the 40th International Conference on Parallel Processing Workshopst. 2011: 279-288.

[4] Huang Y, Tang J, Gu Z, et al. The performance optimization of threaded prefetching for linked data structures [J]. International Journal of Parallel Programming, 2012, 40 (2): 141-163.

[5] 张建勋，古志民．帮助线程预取质量的实时在线评价方法 [J]. 计算机应用, 2017, 37 (1): 114-119.

[6] 张建勋，古志民，胡潇涵，等．面向非规则大数据分析应用的多核帮助线程预取方法 [J]. 通信学报, 2014, 35 (8): 137-146.

[7] Byna S, Chen Y, Sun X. A Taxonomy of data prefetching mechanisms, TR IIT//CS-SCS07-01 [R]. 2007.

[8] 欧国东．基于线程的数据预取技术研究 [D]. 长沙：国防科技大学, 2011.

[9] Kim D, Steve S L, Wang P H, et al. Physical experimentation with prefetching helper threads on intel's hyper-threaded processors [C]// Proc of International Symposium on Code Generation and Optimization. 2004: 27-38.

[10] Song Y, Kalogeropulos S, Tirumalai P. Design and implementation of a compiler framework for helper threading on multi-core processors [C]// Proc of the 14th International Conference on Parallel Architectures and Compilation Techniques. 2005: 99-109.

[11] Lee Jaejin, Jung Changhee, Lim Daeseob, et al. Prefetching with Helper Threads for Loosely Coupled Multiprocessor Systems [J]. IEEE Trans on Parallel and Distributed Systems, 2009, 20 (9): 1309-1324.

[12] Kamruzzaman M, Swanson S, Tullsen D M. Inter-core prefetching for multicore processors using migrating helper threads [C]// Proc of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems, Association for Computing Machinery. 2011: 393-404.

[13] Lu Jiwei, Das A, Hsu W C, et al. Dynamic Helper threaded prefetching on the Sun UltraSparc CMP processor [C]// Proc of the 38th Annual IEEE//ACM International Symposium Microarchitecture. New York: ACM Press, 2005: 93-104.

[14] Luo Yangchun, Packirisamy V, Hsu Weichung, et al. Dynamic performance tuning for speculative threads [C]// Proc of the 36th International Symposium on Computer Architecture. New York: ACM Press, 2009: 462-473.

[15] 裴颂文，张俊格，宁静．梯度学习的参数控制帮助线程预取模型 [J]. 国防科技大学学报, 2016, 38 (5): 59-63.

[16] lllikkal R, Chadha V, Herdrich A, et al. PI-RATE: QoS and Performance Management in CMP Architectures [J]. Newsletter ACM SIGMETRICS Performance Evaluation Review, 2010, 37 (4: 3-10.

[17] Andrew H, Ramesh I, Ravi I, et al. Rate-based qos techniques for cache//memory in CMP platforms [C]// Proc of the 23th Annual International Conference on Supercomputing. 2009: 479-488.

[18] Huang Y, Gu Z, Tang J, et al. Estimating effective prefetch distance in threaded prefetching for linked data structures, international journal of parallel programming [J]. 2012, 40 (5): 465-487.

[19] 黄艳，张启坤，段赵磊，等．基于缓存行为特征的线程数据预取距离控制策略 [J]. 电子与信息学报, 2015, 37 (7): 1633-1638.